МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп‘ютерної та програмної інженерії
Кафедра комп’ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ А.С. Савченко
«___» _____ 20___р

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ «БАКАЛАВР»

Тема: «Гра-Вікторина на базі WEB-технології »

Виконавець: студентка УС-412 Вороніна Євгенія Сергіївна
(студент, група, прізвище, ім'я, по батькові)

Керівник: к. т. н., доцент Клімова Асія Сабирівна
(науковій ступень, вчене звання, прізвище, ім'я, по батькові)

Нормоконтролер: ст. викл. Шевченко О.П. _____
(П.І.Б.) (підпис)

КИЇВ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп‘ютерної та програмної інженерії

Кафедра комп’ютерних інформаційних технологій

Освітній ступінь: Бакалавр

Галузь знань, спеціальність, спеціалізація: 12 “Інформаційні технології”, 122 “Комп’ютерні науки”, “Інформаційні управляючі системи та технології”

ЗАТВЕРДЖУЮ
Завідувач кафедри

А.С. Савченко
“_____”_____ 2021 р.

ЗАВДАННЯ

на виконання дипломного проекту студента

Вороніна Євгенія Сергіївна

(прізвище, ім’я, по батькові)

1.Тема проекту: «Гра-Вікторина на базі WEB-технології» затверджена наказом ректора № 636/ст. від 22.04.2021р.

2. Термін виконання роботи: з 10.05.2021 по 12.06.2021р.

3. Вихідні дані до роботи: розробка гри-вікторини на базі WEB-технології.

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): вступ, аналітичний огляд і постановка завдання, розробка та опис програмного рішення, висновки.

5. Перелік обов’язкового графічного матеріалу: загальний перелік існуючих систем та діаграми описів процесів роботи системи.

КАЛЕНДАРНИЙ ПЛАН

	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1	Аналіз літератури та джерел за темою дипломного проекту.	10.05.2021р. – 12.05.2021р.	
2	Розробка та затвердження плану дипломного проекту.	13.05.2021р.	
3	Проведення консультації з науковим керівником щодо створення першого розділ.	14.05.2021р.	
4	Аналітичний огляд і постановка задачі.	15.05.2021р. – 18.05.2021р.	
5	Порівняльний аналіз існуючих систем	19.05.2021р. – 22.05.2021р.	
6	Опис функціональних можливостей	23.06.2021р. – 27.05.2021р.	
6	Розробка гри-вікторини	28.05.2021р. – 04.06.2021р.	
7	Висновки та оформлення пояснювальної записки дипломного проекту.	05.06.2021р. – 08.06.2021р.	
8	Підписання необхідних документів у встановленому порядку.	09.06.2021р. – 10.06.2021р.	
9	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломного проекту	11.06.2021р. – 12.06.2021р.	

Дата видачі завдання 10 травня 2021 року

Керівник дипломної роботи _____ Клімова А.С.
(підпис)

Завдання прийняв до виконання _____ Вороніна Є.С.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Гра-Вікторина на базі WEB-технології» містить: 60 сторінки, 13 рисунків, 6 таблиць, 15 літературних джерел.

Акутальність роботи полягає у актуалізації проблеми ефективності використання сучасних WEB-технологій при реалізації WEB-інтерфейсу користувача, а також проблема організації навчального процесу з використанням новітніх технологій.

Об'єкт дослідження: веб застосунок для проведення ігрових та навчальних комунцікацій.

Предмет дослідження: веб застосунки.

Мета роботи: розробка гри-вікторини на основі WEB.

Методи дослідження, технічні та програмні засоби: розробка, порівняльний аналіз, синтез, абстрагування, узагальнення, обробка літературних джерел.

Отримані результати та їх новизна: Пропонований варіант рішення дозволяє успішно інтегрувати стек сучасних WEB-технологій та застосувати його в розробці програмного комплексу для провдення занять у ігровій формі, сприяє підвищенню зацікавленості учнів в навчальному процесі та автоматизує процес процес проведення котрольних та заліків.

WEB, ГРА, ВІКТОРИНА, ВЕБ-ДОДАТОК, JAVASCRIPT, КЛІЄНТ-СЕРВЕР, ПОТІК ДАНИХ, РОЗРОБКА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

WEB - World Wide Web

HTML - HyperText Markup Language

CSS - Cascading Style Sheets

PWA - Progressive Web Applications

HTTPS - HyperText Transfer Protocol Secure

SW - Service Worker

API - Application Programming Interface

URL - Uniform Resource Locator

MVC - Model-view-controller

СУБД - Систе́ма управління ба́зами да́них

БД - База даних

ВСТУП

Вплив глобальної комп'ютерної мережі Internet на сучасний світ не має історичних аналогів. Його сьогоднішній день - це початок епохи електронного заповнення усіх сфер людського життя, це основа нової філософії і нового способу мислення.

Web-технології повністю перевернули уявлення про роботу з інформацією, та й з комп'ютером взагалі. Виявилося, що традиційні параметри розвитку обчислювальної техніки - продуктивність, пропускна здатність, ємність запам'ятовуючих пристроїв - не враховували головного "вузького місця" системи - інтерфейсу з людиною. Застарілий механізм взаємодії людини з інформаційною системою стримував впровадження нових технологій і зменшував вигоду від їх застосування. І тільки коли інтерфейс між людиною і комп'ютером був спрощений до природності сприйняття звичайною людиною, послідував безпрецедентний вибух інтересу до можливостей обчислювальної техніки.

Сучасні технології інтегруються в різні сфери нашого життя щодня. Одна з таких сфер – освіта, що розвивається щодня, розробляються нові методи для покращення рівню освіти. Саме тому інтеграція сучасних технологій в освітній процес сприяє зміні підходу до навчання, збільшенню об’єму поглинутих знань, використанню нові технології для самореалізації у не доступних раніше сферах.

Багато освітніх методів створені завдяки використанню комп’ютерів, смартфонів, планшетів. З вимушеним переходом на дистанційне навчання, завдяки освітнім платформам та застосункам, що дозволяють користувачу набувати знань по підготовленим програмам, можливості збільшуються. За рахунок щохвилинної доступності до навчального матеріалу та відсутності фіксованого часу для навчання, користувачі мають можливість навчатись у будь-який час. Це зробило освітній процес зручним та продвинутим. Відповідно це сприяє навчальному процесу та провокує освітні програми до розвитку та вдосконалення. Нові формати представлення інформації дозволяють експериментувати з створенням персональних навчальних підходів, що сприяє підвищенню продуктивності навчання та збільшує

кількість засвоєного матеріалу. Одним з таких форматів є освітні *web*-платформи, що надають можливість охопити всі вікові категорії, персоналізуючи освітній матеріал під вік користувача та подаючи навчання в різних формах: лекцій, відеоматеріалів, ігрових застосунків.

Головними задачами роботи є:

1. вивчення й аналіз існуючих систем, які дозволяють впроваджувати в навчальний процес комп'ютерних технологій;
2. порівняльний аналіз конкурентних систем та практик;
3. розробка ігрового додатку для проходження вікторин на базі веб застосунку.

У роботі актуалізується проблема ефективності використання сучасних *WEB*-технологій при реалізації *WEB*-інтерфейсу користувача, а також проблема організації навчального процесу з використанням новітніх технологій.

Пропонований варіант рішення дозволяє успішно інтегрувати стек сучасних WEB-технологій та застосувати його в розробці програмного комплексу для провдення занять у ігровій формі, сприяє підвищенню зацікавленості учнів в навчальному процесі та автоматизує процес процес проведення контрольних та заліків.

РОЗДІЛ 1
ЗАГАЛЬНА ХАРАКТЕРИСТИКА ЛОКАЛЬНОЇ МЕРЕЖІ

1.1. Основні поняття WEB

Web-вузол (Web-сайт) - це комплекс Web-сторінок та інших ресурсів, об'єднаних відповідно до змістом, розміщених на одному домені. Web-сайт - це загальне інформативне джерело. Мережа Internet складається з великого числа з'єднаних між собою комп'ютерів, маршрутизаторів та іншого, необхідного для правильної роботи. Кожен елемент мережі Internet (вузол) має неповторний описателем - IP-адресу. Знаючи IP-адреса вузла, є можливість спробувати під'єднатися до нього, а маючи невеликі навички можна з'ясувати кому, ця адреса належить і в якому знаходиться регіоні світу. IP-адреси прийнято записувати у вигляді чотирьох груп цифр, розділених крапками, наприклад 192.168.100.003 або 10.10.0.123.

Запам'ятати адреси всіх часто відвідуваних сторінок практично неможливо, для цього в мережі Internet існують спеціальні сервера DNS (Domain Name System), на яких виконується зберігання всіх списків зіставлення IP-адресу і символьних імен. Завдяки цьому, до серверів, користувач завжди переходить за потрібною IP-дресою, набравши в браузері тільки ім'я сторінки. Після цього, як ми ввели в рядок браузера ім'я необхідної сторінки, браузер автоматично отримує з DNS IP-адресу потрібного сервера і посилає за цією адресою спеціальний запит на отримання сторінки (HTTP-запит). Працююча на сервері спеціальна програма (т.зв. Web-сервер) обробляє цейзапит і відправляє назад в браузер необхідну сторінку.

Очевидно, що всі дії по відображенню сторінки можна розділити на дві категорії: що виконуються на стороні клієнта (код клієнтський або front-end) і на

стороні сервера (код серверний або back-end). При цьому сервер зовсім нічого не знає про поточний стан клієнта, а клієнт - про стан сервера.

При розробці алгоритмів обміну потрібно завжди знати про це і своєчасно пересилати потрібні дані, що описують поточний стан або потрібні дії.

Залежно від середовища використання розрізняються і засоби реалізації частин. На стороні клієнта використовується зазвичай тільки HTML, JavaScript (AJAX), CSS, Flash.

1.2. Аналіз існуючих систем

Сучасні технології інтегруються в різні сфери нашого життя щодня. Одна з таких сфер – освіта, що розвивається щодня, розробляються нові методи для покращення рівню освіти. Саме тому інтеграція сучасних технологій в освітній процес сприяє зміні підходу до навчання, збільшенню об’єму поглинутих знань, використанню нові технології для самореалізації у не доступних раніше сферах.

Багато освітніх методів створені завдяки використанню комп’ютерів, смартфонів, планшетів. З появою віддаленого навчання, завдяки освітнім платформам та застосункам, що дозволяють користувачу набувати знань по підготовленим програмам, можливості збільшуються. За рахунок щохвилинної доступності до навчального матеріалу та відсутності фіксованого часу для навчання, користувачі мають можливість навчатись у будь-який час. Це сприяє навчальному процесу та дозволяє розвиватись освітнім методам у новому руслі.

Udemy

“Udemy” — це освітня Web-система, яка базується на курсах із великого різноманіття тем. ЇЇ важливою характеристикою можна назвати те, що майже кожен може розробити свій курс. Це також є своєрідним недоліком, так як ніхто не перевіряє корисність курсу та його навчальну пригодність. Ще одним важливим моментом є те, що проходження курсів на цій системі ніяк юридично не підтверджується;

Coursera

“Coursera” — це платформа, на якій університети створюють свої курси. На відміну від попередньою платформи, тут після проходження курсу видається

сертифікат, але зв’язок із розробниками курсу досить обмежений. Але це пояснюється віддаленістю викладачів. Також певні курси не мають практичних уроків, інформація виклалається у вигляді лекцій.

Також недоліком цих систем є їх громіздкість. На цих платформах дуже багато різноманітних курсів, тому досить складно знайти потрібний освітній курс.

1.3. Progressive Web Applications

PWA - це новий підхід розробки Web додатків, запропонований Google. В основі нього лежать звичайні і всім відомі технології: HTML, CSS і Javascript. Але додаючи нові технології поверх цієї основи і, слідуючи 10 основним концепту запропонованими Google, можна зробити передовий прогресивний додаток (рис.1.1).

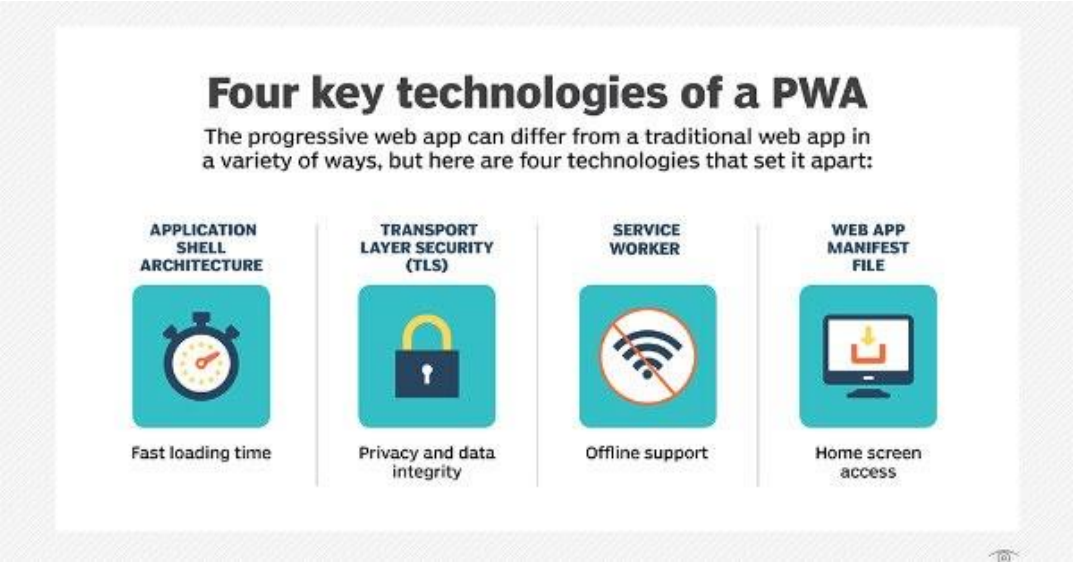

Рис.1.1. Чотири основні технології PWA

Нижче представлені приклади PWA-додатків:

- Safe - використовує HTTPS для запобігання атаки "людина посередині" і цілісності контенту;
- Progressive - працює для всіх користувачів незалежно від вибору браузера;
- Responsive - розміщення на будь-якому форм-факторі: десктоп,

смартфон, планшет, годинник, і будь-яке інше, що ще не існує;

- Connectivity-independent - використовуючи технологію Service Worker, дає доступ до можливостей додатка при відсутності або при повільному інтернет з'єднанні;
- App-like - відчувається як нативний додаток для користувача з взаємодіями в стилі додатку і навігації, тому що вони побудовані на моделі оболонки програми;
- Fresh - постійно останньої версії, завдяки процесу оновлення, котрі використовують SW;
- Discoverable - Є які можуть бути ідентифіковані як «додаток» завдяки маніфесту W3C і реєстрації Service Worker;
- Re-engageable - робить повернення користувачів простим, використовуючи Push повідомлення;
- Installable - дозволяє користувачам зберігати додаток на робочий стіл без складнощів пов'язаних з використанням магазину додатків;
- Linkable - додає можливість ділитися додатком за допомогою звичайного посилання, не використовуючи складні процеси установки.

Safe

Оскільки прогресивні веб-додатки можуть передавати конфіденційні дані через нативне API і Service Worker, рекомендується використовувати прогресивні веб-додатки через HTTPS-з'єднання.

Progressive

Прогресивність – мається на увазі техніка, яка використовується для поліпшення контенту та можливостей додатка, наскільки це дозволяє браузер або інтернет-з'єднання, але основні функції або, принаймні, правильні повідомлення про помилку повинні бути доступні для всіх.

Ядром прогресивного веб-додатку є новий API браузера Service Worker, який поки недоступний у всіх браузерах, але додаток все одно може працювати навіть без нього.

Іншим прикладом нового API браузера є доступ до камери на пристрої через API getUserMedia. Щоб все люди могли використовувати цю функціональність

програми, необхідно охопити всі конкретні префікси API, що використовуються різними браузерами.

Responsive

Чутливий дизайн означає, що призначений для користувача інтерфейс відображається таким чином, який відповідає розмірам пристрою. Додаток має мати гнучкий дизайн і працювати з кожним розміром екрану. Чутливий дизайн дуже важливий, так як прогресивні веб-додатки націлені на мобільні пристрої. Такий дизайн може бути досягнутий з використанням концепцій плаваючої сітки, гнучких зображень і медіа запитів CSS3.

Connectivity independent

Технологія Service Worker дозволяє вашому додатку працювати без підключення до Інтернету. Service Worker - клієнтський проксі-сервер, написаний на JavaScript, який допомагає керувати кешом розробнику і описувати стратегію реагування на запити ресурсів. Попередньо копіюючи основні ресурси, додаток може усунути залежність від мережі, забезпечуючи миттєвий і надійний UX для ваших користувачів.

Деякі додатки як і раніше у великій мірі залежать від динамічного контенту онлайн серверів, але навіть ці додатки можуть кешувати елементи призначеного для користувача інтерфейсу і швидше запускати і навіть відображати деякі кешированниє дані з попереднього взаємодії.

Щоб використовувати Service Worker, розробник повинен створити файл JavaScript, який буде зареєстрований як Service Worker і прослухати як мінімум дві події, "встановити" і "витягнути" і реалізувати основну логіку.

З зареєстрованим Service Worker все файли програми можуть бути збережені в кеші браузера і доступні для користувача, коли він повернеться. Відповіді з даними і змінними додатки можуть бути збережені в localStorage або IndexDB, де це можливо.

Service Worker - абсолютно нова технологія і ще не підтримується всіма веб-браузерами, але вже має достатню підтримку, щоб використовувати у внутрішній розробці (рис. 1.2)

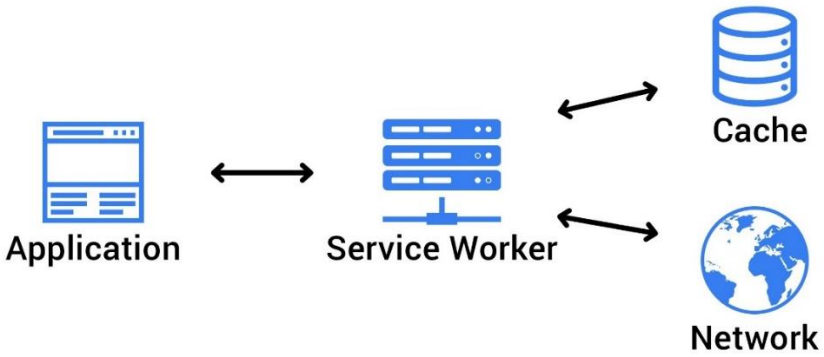

Рис. 1.2. Принцип роботи Service Worker

App-like

Концепцію дизайну, яку ще називають архітектурою App-shell, рекомендується використовувати при побудові користувальницького інтерфейсу Progressive web application. Ця концепція відокремлює додаток від оболонки і вмісту (рис. 1.3).

Shell - це статичні частини програми, необхідні для показу контенту. Коли додаток кешується, необхідно кешувати всю оболонку на пристрої. Кешована оболонка додатку гарантує, що навіть якщо немає підключення до Інтернету, і додаток з кешуватися динамічним вмістом не завантажить хоча б знайомий користувальницький інтерфейс, і користувачеві не буде надано порожній екран або повідомлення про помилку з'єднання за замовчуванням. Вміст є видимим всередині оболонки. Сам контент також може бути кешуватися, але він сильно залежить від типу програми.

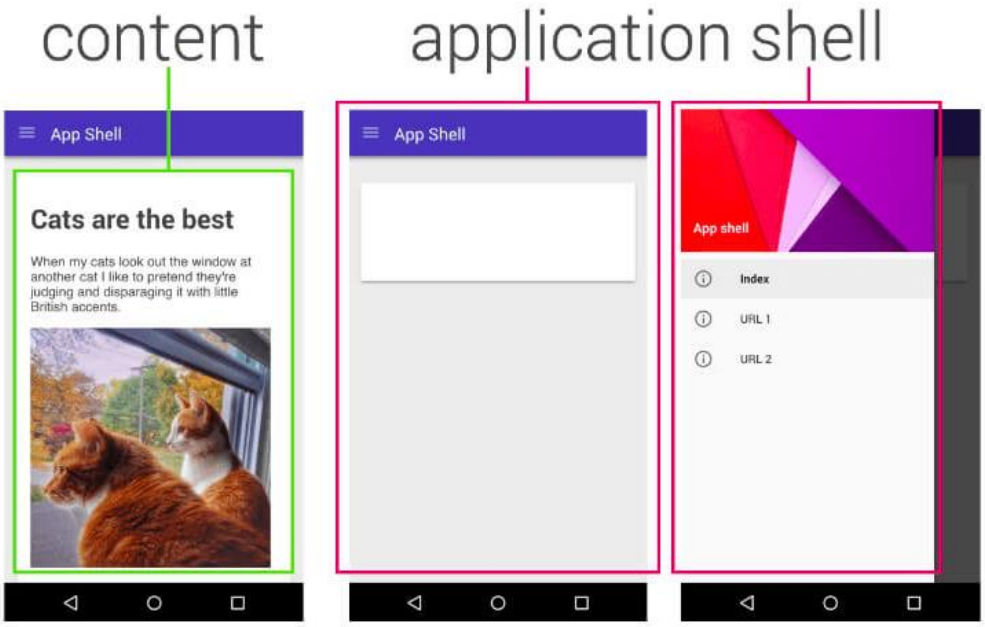

Рис. 1.3. Поняття App-shell

Fresh

Вміст додатку і кешований контент завжди будуть завантажуватися з локального сховища після його кешування (рис. 1.4) . Однак браузер перевіряє, чи був змінений файл Service Worker з останнього відвідування, і якщо це так, нова версія Service Worker встановлюється для завантаження наступної сторінки. Цей новий Service Worker може знову кешувати всі ресурси і видаляти старі кешовані дані.

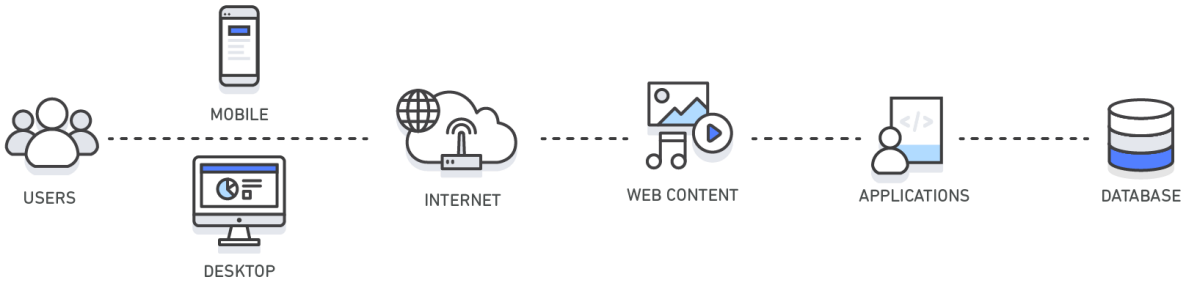

Рис. 1.4. Алгоритм кешування даних

Discoverable

Використання Web Manifest дозволяє розробнику надавати різну інформацію про веб-додаток і додатково описувати як воно повинно відображатися. Web Manifest також дозволяє встановлювати додаток на головний екран і відкриватися в окремому вікні браузера.

Web Manifest - це файл JSON, заданий в заголовку HTML-документа index сторінки додатка.

Re-engageable

Прогресивний додаток може використовувати push-повідомлення завдяки Service Worker, API Push і Notifications API. Завдяки цим API-інтерфейсам веб-додаток може отримувати повідомлення з сервера. Ці повідомлення можуть відображатися як повідомлення користувачу і повідомляти йому про оновлення або заплановані події. Це спрощує повторну взаємодію з користувачем і повертає його в додаток.

Installable

Будь-який веб-сайт можна зберегти на головному екрані Android / Windows / Mac з Chrome, використовуючи кнопку «Додати на головний екран» в меню.

Однак Chrome дозволяє запитувати у користувача спливаюче вікно пропонуючи йому зберегти додаток на головному екрані. Це спливаюче вікно недоступне розробнику, але браузер автоматично відобразить спливаюче вікно, якщо додаток виконає наступні вимоги.

Веб-маніфест повинен бути представлений і відповідати специфікації. Додаток має мати дійсний і встановлений Service Worker, а додаток має обслуговуватися через HTTPS.

Якщо користувач підтверджує діалог, додаток негайно зберігається на головному екрані і може бути зручно доступно в наступний раз, коли користувач захоче його відкрити.

Linkable

Оскільки прогресивні веб-додатки доступні для всіх, у кого є браузер, вони можуть передаватися безпосередньо через URL. Також контент і окремі сторінки додатку повинні бути пов'язані один з одним, щоб забезпечити можливість спільного використання і можливість повторного відкриття програми на тому ж місці.

Прогресивні веб-додатки - це не нові технології або рамки, це новий підхід до створення веб-додатків, які ведуть себе як нативні.

Google докладає великі зусилля в наданні навчальних посібників і підтримки цього методу. Варто відмітити, з більшою підтримкою браузерів і платформ це

може бути майбутнє мобільної мережі. Додатки також можуть скористатися перевагами технологій, згаданих вище, навіть на робочому столі, де також дуже корисні швидкі і автономні доступні веб-додатки. Особливо для кешування використовувати Service Worker.

1.4. Висновки до розділу

В даному розділі проаналізовано предметна область, виділені основні проблеми і завдання, які пропонується вирішити за допомогою розробленого Web-додатки. Визначено вимоги, як до самого Web-додатком, так і до програмно-технічним засобам для коректної роботи Web-додатку.

Сучасні технології інтегруються в різні сфери нашого життя щодня. Одна з таких сфер – освіта, що розвивається щодня, розробляються нові методи для покращення рівню освіти. Саме тому інтеграція сучасних технологій в освітній процес сприяє зміні підходу до навчання, збільшенню об’єму поглинутих знань, використанню нові технології для самореалізації у не доступних раніше сферах.

Багато освітніх методів створені завдяки використанню комп’ютерів, смартфонів, планшетів. З появою віддаленого навчання, завдяки освітнім платформам та застосункам, що дозволяють користувачу набувати знань по підготовленим програмам, можливості збільшуються. За рахунок щохвилинної доступності до навчального матеріалу та відсутності фіксованого часу для навчання, користувачі мають можливість навчатись у будь-який час. Це сприяє навчальному процесу та дозволяє розвиватись освітнім методам у новому руслі.

РОЗДІЛ 2
ВИБІР ТА ОБГРУНТУВАННЯ ПРОГРАМНИХ ТА АПАРАТНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Вибір архітектури програмного комплексу

Для реалізації задачі було прийняте рішення використати наступну архітектуру: веб-застосунок, мобільний застосунок, сервер і база даних. Схема обраної архітектури зображена на рис. 2.1.

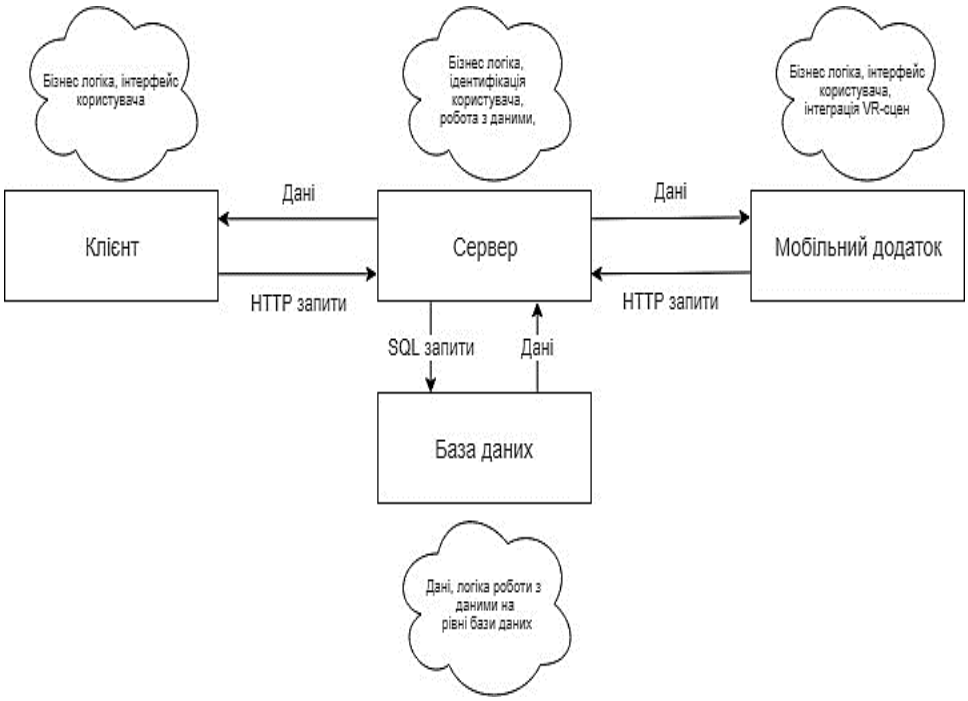

Рис. 2.1. Архітектура програмного комплексу

Сервер – основний компонент обраної архітектури. Він містить у собі основну бізнес-логіку та забезпечує доступ до бази даних. Для надання персонального доступу до додатку відбувається ідентифікація користувача за допомогою серверу.

Ланкою, що слугує зв’язком між базою даних та користувачем є сервер. Побудована архітектура наступним чином щоб попередити можливе використання даних не за призначенням та їх пошкодження. Для того, щоб користуватися програмою, користувач має бути авторизованим у системі, тому дана логіка реалізується на рівні серверу.

При користуванні освітньою системою, користувач взаємодіє з клієнтським додатком, веб-сайтом в даному випадку. Саме тому для користувача реалізовано інтерфейс, завдяки якому відбувається налаштування програми та перегляд курсів. А ще на користувацькому рівні відбувається первинна обробка даних до того як відправити на сервер і опрацювати результатів, отриманих від сервера. Також на цьому рівні відбувається перший етап автентифікації користувача, який потрібний для обмеження неконтрольованого доступу до програми.

Під час проходження практичної частини курсів, користувач взаємодіє веб застосунком. У ньому реалізований інтерфейс автентифікації.

Сервер зберігає дані для подальшого використання на рівні бази даних. Цілісність даних забезпечується за допомогою зовнішніх зв’язків та ключів. Частину бізнес-логіки можна реалізувати на рівні самої бази даних, якщо вона не потребує використання інших джерел.

Подальший опис етапів створення кінцевого додатки буде заключатися на ітераційному підході, в основі якого лежить з поступове ускладнення і деталізація первісної структури програми.

В основі розроблюваної системи лежить архітектура «клієнт-сервер», в якій мережеве навантаження розподілене між постачальниками послуг (сервісів), які називаються серверами і замовниками послуг, так званими клієнтами.

Клієнтом є браузер користувача, де за допомогою DOM і JavaScript забезпечується взаємодія користувача з даними. Використання JavaScript само по собі не є обов'язковим, однак функціонування більшості Web-додатків без JS не представляється можливим. Як середовище взаємодії клієнта з сервером використовується мережу Internet (рис. 2.2).

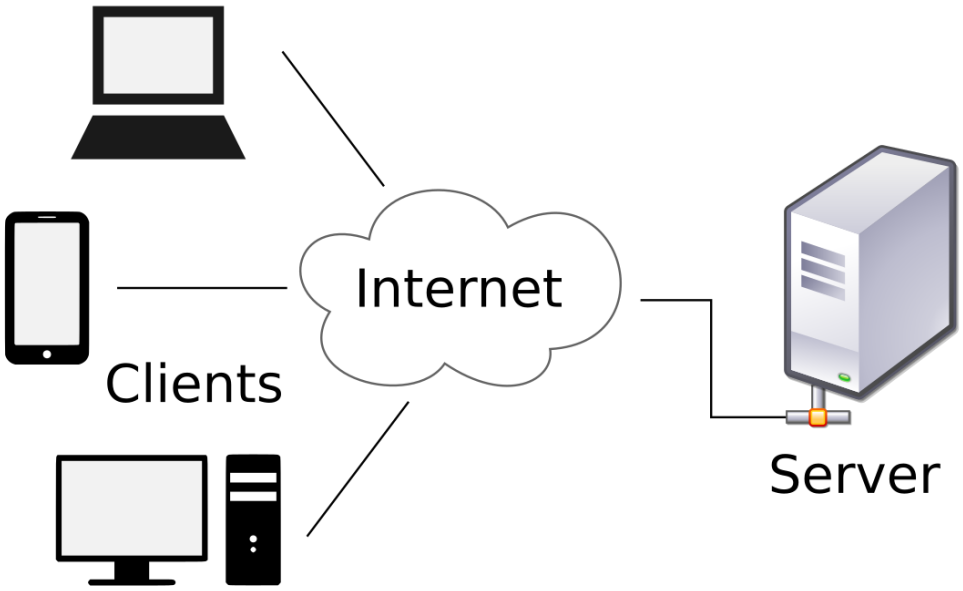

Рис. 2.2. Схема роботи архітекутури «Клієнт-сервер»

Також для вирішення поставленого завдання можна використовувати архітектуру «Трирівнева архітектура», проте в даному випадку подібний підхід буде надлишковий з причини відсутності великої фрагментованої бізнес-логіки додатка. Програмний застосунок для освітньої web-системи містить у собі двох головних акторів – користувач системи і адміністратор системи.

Основними перевагами архітектури «клієнт-сервер» є:

- можливість, в більшості випадків, розподілити функції обчислювальної системи між декількома незалежними комп'ютерами в мережі. Це дозволяє спростити обслуговування обчислювальної системи. Зокрема, заміна, ремонт, модернізація або переміщення сервера, не зачіпають клієнтів;
- всі дані зберігаються на сервері, який, як правило, захищений набагато краще за більшість клієнтів. На сервері простіше забезпечити контроль повноважень, щоб дозволяти доступ до даних тільки клієнтам з відповідними правами доступу;
- дозволяє об'єднати різні клієнти. Використовувати ресурси одного сервера часто можуть клієнти з різними апаратними платформами, операційними системами і т.п.

Основним недоліком архітектури «клієнт-сервер» (рис 2.3) є використання централізованої системи, непрацездатність основного сервера може зробити непрацездатною весь Web-додаток.

До завдань сервера БД входить:

• обслуговування запитів на маніпуляції з даними на основі мови SQL;

• обслуговування БД;

• забезпечення цілісності даних;

• надання утиліт для адміністративного управління СУБД.

Клієнтська частина програми повинна підтримувати такі техногії:

• доступ до мережі Internet;

• можливість роботи по протоколу HTTP / 1.1;

• висновок інформації;

• підтримка пристроїв введення даних.

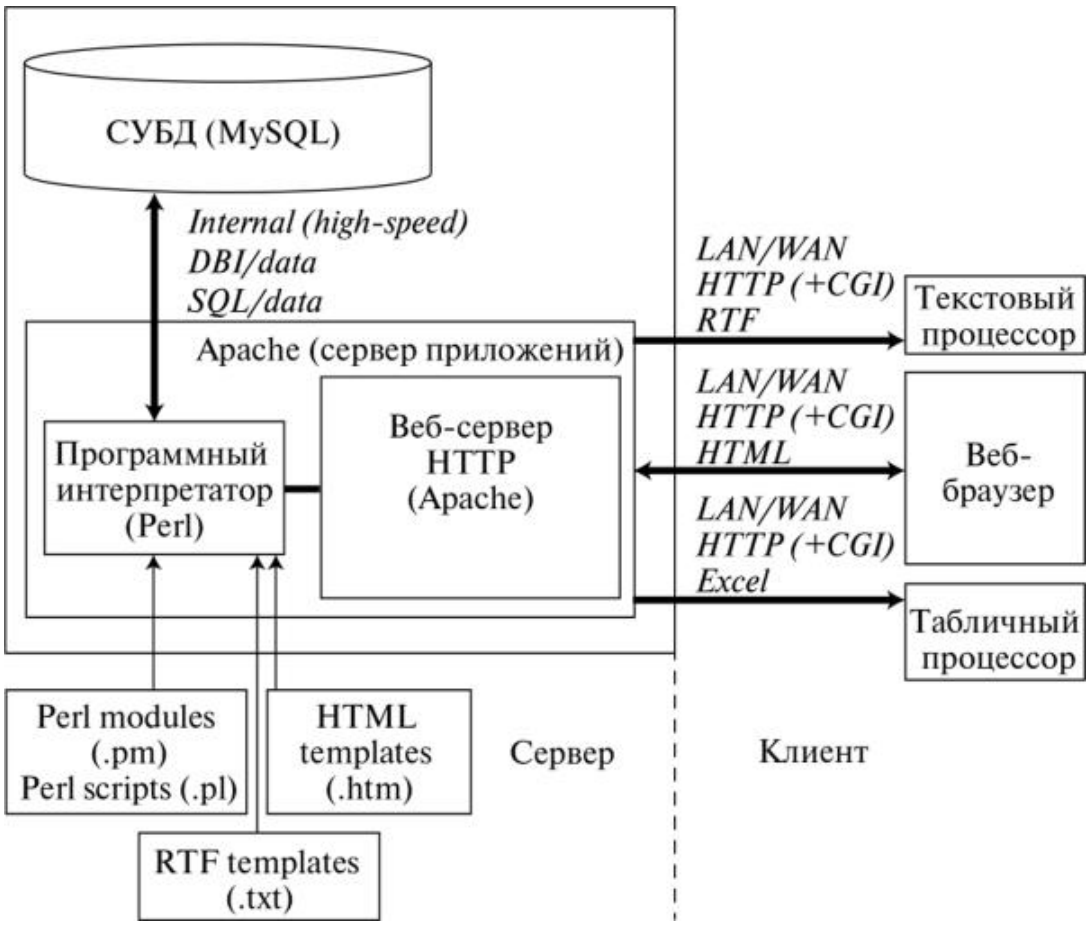

Рис. 2.3. Розгорнута архітектура «Клієнт-сервер»

Model-View-Controller

MVC - схема поділу даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: модель, уявлення і контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно.

MVC ділить інтерактивне додаток на три різні області: обробка, висновок і введення. Модель виконує роль обробки, уявлення забезпечує візуальну зворотний зв'язок, а контролер обробляє вхід або взаємодія користувача.

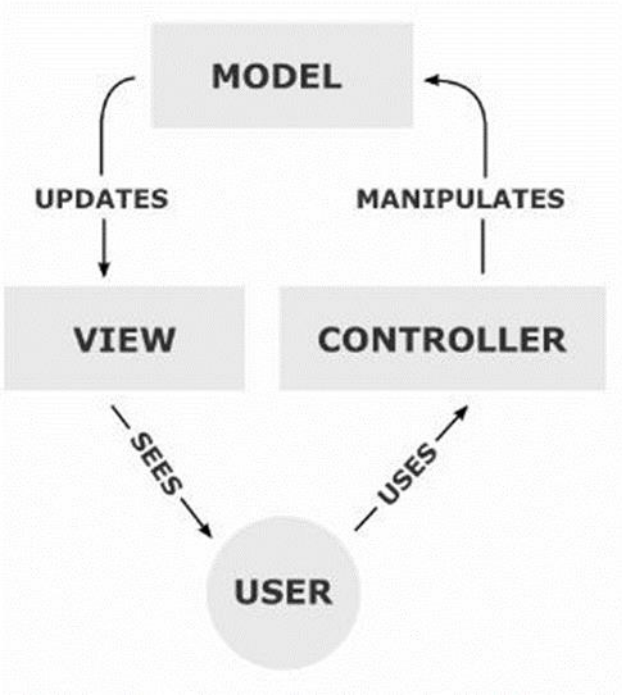

Рис. 2.4. Model-View-Controller

Model-View-Controller являє собою невізуальний об'єкт, що містить всі дані і поведінку, відмінні від того, яке використовується для його уявлення користувачеві. Model-View-Controller представляє дані і стан програми та логіки домена, які її маніпулюють. Модель надає інтерфейс для інших компонентів для доступу до даних. Щоб максимізувати инкапсуляцию даних і повторне використання коду, модель повністю не знає про інших компонентах і не залежить від них.

Model-View-Controller може бути як простий цілочисельний змінної, так і складною, як корпоративний веб-сервіс, що обслуговує дані JSON або XML, і

зведений будинок із використанням багатошарової або навіть багаторівневої архітектури, інкапсулюють бізнес-правила і вимоги до стабільності.

View

Візуальний показ програми користувачеві виконується за допомогою відображень. Відображення не знають, які дані відображаються користувачеві, вони тільки представляють його. Робота перефіріних пристроїв передає дії користувача, це може бути рух миші, штрихи клавіатури або жести торкання до контролера. Відображення зазвичай вкладені, так що у представленні містяться субвідображення всередині нього. Вид згори цієї ієрархії може відповідати за загальні аспекти, такі як управління вікном, в якому виконується додаток. У випадку з MVC це реалізовано за допомогою шаблону Компоновщик, а також реалізовано безліч сучасних інструментів і інтерфейсів для користувача інтерфейсу, що використовують цей патерн**.**

Controller

Контролер - це те, що пов'язує представлення і моделі в MVC. Контролер приймає вхід користувача як події з представлення і приймає інтелектуальні рішення для представлення. Цей механізм подій виконується за допомогою шаблону стратегії. Шаблон стратегії робить контролери мінливими і може використовуватися, наприклад, для забезпечення редагованої і незмінної поведінки в одній і тій же формі.

Flux - це архітектура додатків, яка використовується Facebook для створення клієнтських веб-додатків. Вона доповнює складові компоненти представлення React допомогою односпрямованого потоку даних. Це скоріше шаблон, ніж формальна структура і ви можете відразу почати використовувати Flux без великої кількості нового коду.

Додатки Flux мають три основні частини: диспетчер (dispatcher), сховище (store) і представлення (view). Їх не слід плутати з Model-View-Controller. Контролери існують в додатку Flux, але вони є представленнями, які часто зустрічаються у верхній частині ієрархії, які витягають дані зі сховищ і передають ці дані своїм нащадкам. Крім того, творці дій (action) - допоміжні методи диспетчера - використовуються для підтримки семантичного API, який описує всі зміни, які можливі

в додатку. Корисно подумати про них як про четверту частину циклу поновлення потоку.

Flux уникає MVC на користь односпрямованого потоку даних. Коли користувач взаємодіє з представленнями React, представлення передає дію через центральний диспетчер в різні сховища, в яких зберігаються дані і бізнес-логіка програми, що оновлює всі переглянуті види. Особливо гостро це працює із декларативним стилем програмування React, який дозволяє поновлення сховища відправляти без вказівки способу переходу між станами.

Управління інвертується сховищами: сховища приймають поновлення і погоджують їх у міру необхідності, а не в залежності від чогось зовнішнього, щоб оновлювати свої дані узгодженим чином. Ніщо за межами сховища не має жодного уявлення про те, як він керує даними, які підтримуються вашим, допомагаючи зберегти чіткий поділ проблем. У сховищах немає прямих методів настройки, таких як setAsRead (), але замість цього є тільки один спосіб отримання нових даних в їх автономний світ - зворотний виклик, який вони реєструють в диспетчері.

Структура і потік даних. Дані в додатку Flux надходять в одному напрямку(рис.2.5)

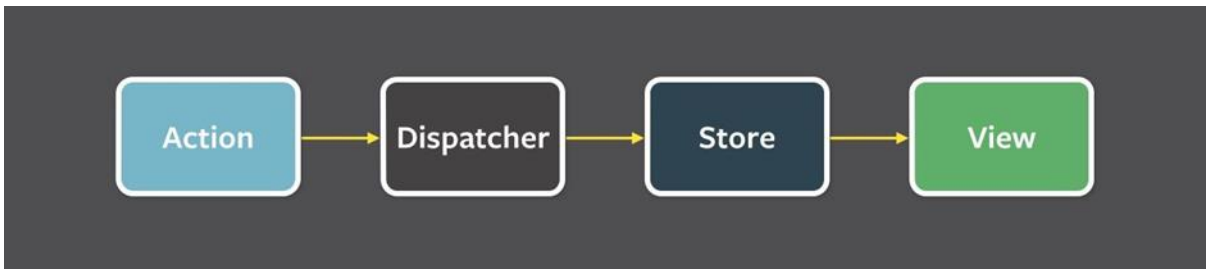

Рис. 2.5. Потоки даних в додатку Flux

Односпрямований потік даних є центральним для Flux, і наведена вище діаграма повинна бути основною ментальної моделлю для програміста працює з Flux. Диспетчер, сховища і подання є незалежними вузлами з різними входами і виходами. Дії - це прості об'єкти, що містять нові дані і властивість ідентифікаційного типу.

Представленя можуть призводити до поширення нової дії через систему у відповідь на взаємодію користувача (рис. 2.6).

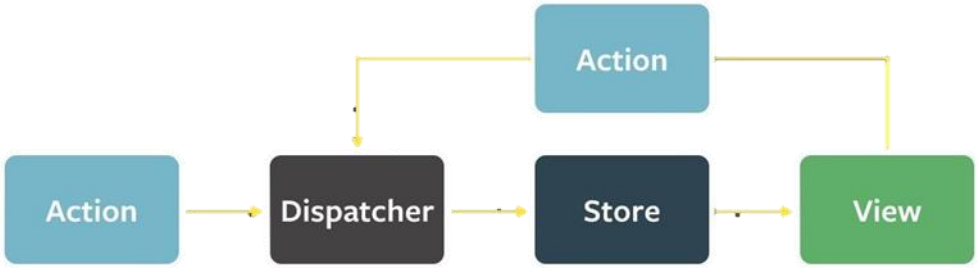

Рис. 2.6. Поширення нового Action у відповідь на взаємодію користувача

Всі дані проходять через диспетчер в якості центрального концентратора. Дії надаються диспетчеру в методі створення дії і найчастіше відбуваються з користувацьких взаємодій з уявленнями. Потім диспетчер викликає зворотні виклики, які були зареєстровані в сховищах, відправляючи дії в усі сховища.

У межах своїх зареєстрованих зворотних викликів сховища відповідають на те, які дії мають відношення до стану, який вони підтримують. Потім сховища видають подія зміни, щоб попередити контролери-уявлення про те, що відбулася зміна стану даних.

Контролери переглядають ці події і витягають дані зі сховищ в обробнику подій. Контролери-уявлення викликають свій власний метод setState (), викликаючи повторний рендеринг самих себе і всіх їх нащадків в дереві компонентів.

Ця структура дозволяє нам легко міркувати про наш додатку таким чином, який нагадує функціональний реактивне програмування або, більш конкретно, програмування потоками даних або програмування на основі потоку, де дані проходять через додаток в одному напрямку. Стан додатку існує тільки в сховищах, дозволяючи різним частинам додатки залишатися сильно роздільними. Якщо між сховищами існують залежності, вони зберігаються в суворій ієрархії з синхронними оновленнями, керованими диспетчером.

Ми виявили, що двосторонні прив'язки даних привели до каскадних оновлень, коли зміна одного об'єкта привела до зміни іншого об'єкта, що також могло викликати додаткові оновлення. У міру зростання додатків ці каскадні поновлення все

складніше передбачити, що зміниться в результаті взаємодії з користувачем. Коли оновлення можуть змінювати тільки дані за один раунд, система в цілому стає більш передбачуваною.

Мова програмування JavaScript сконструйована компанією Netscape з метою формування інтерактивних HTML-документів. Це об'єктно-орієнтована мова розробки вбудованих додатків, які працюють як на стороні клієнта, так і сервера. Синтаксис мови дуже схожий на синтаксис мови Java - тому його часто називають Java-подібним. Клієнтські програми виконуються браузером перегляду Web-документів на комп'ютері користувача, а серверні додатки виконуються на сервері.

При розробці обох типів додатків застосовується єдиний компонент мови, іменований ядром і містить визначення стандартних об'єктів і конструкцій (змінні, функції, основні об'єкти і засоби LiveConnect взаємодії з Java-апплетами), і відповідні компоненти доповнень мови, що мають характерні для кожного типу додатків визначення об'єктів .

Клієнтські програми безпосередньо вбудовуються в HTML-сторінки і інтерпретуються браузером згідно відображенням частин документа в його вікні. Серверні додатки з метою підвищення продуктивності заздалегідь компі- люються в проміжний byte-code.

Основні галузі використання мови JavaScript при створенні інтерак- тивних HTML-сторінок:

- динамічне формування документа за допомогою сценарію;
- оперативний контроль правдивості заповнюваних полів пользовате- лем форм HTML аж до передачі їх на сервер;
- створення динамічних HTML-сторінок спільно з каскадними таблицями стилів і об'єктної моделлю документа;
- взаємодія з користувачем при вирішенні "локальних" завдань, що вирішуються додатком JavaScript, вмонтованому в HTML-сторінку.

2.2. Функціональні вимоги для додатку

Метою даної випускної кваліфікаційної роботи є розробка Web-додатку, що дозволяє надавати інформаційно-навчальний матеріал та подавати його в інтерактивному форматі гри-вікторини.

Основними завданнями програми є: надавати інформаційно-навчальний матеріал, підтримка інтерактивного формату, можливість отримати результати роботи без затримок та зберігти результати у кабінеті користувача, а також забезпечення доступу через мережу Internet до Web-ресурсу.

Проаналізувавши предметну область і оцінивши ресурси підприємства, було прийнято рішення розробити Web-додаток. На сьогоднішній день технології дозволяють створювати Web-додатки високої складності з різним функціоналом.

Переваги Web-додатків:

- програмний код Web-додатки виконується на віддаленому сервері, не використовуючи ресурсів машини користувача;
- відсутність необхідності установки на машину користувача будь-яких компонентів додатка;
- відсутність проблем з оновленням і підтримкою старих версій програм;
- забезпечення мобільності користувачів.

Безумовно, у Web-додатків є істотні недоліки:

- необхідність підключення користувача до мережі (локальної або глобальної) для доступу до сервера;
- швидкість роботи програми залежить від швидкості передачі даних між клієнтом і сервером і завантаженості сервера, яка тим вище, чим більше користувачів одночасно звертаються до сервера.

Сьогодні Web-додатки широко використовуються на підприємствах з розвиненою корпоративною мережею, так як супровід і оновлення таких додатків обходиться набагато дешевше і не вимагає великих витрат часу.

2.3. Архітектура програми

Додаток складається з двох основних частин: Веб-додаток і Electron додаток, яке відповідає за генерацію файлів, які будуть нативної запускатися в необхідної середовищі: .exe, .dmg.

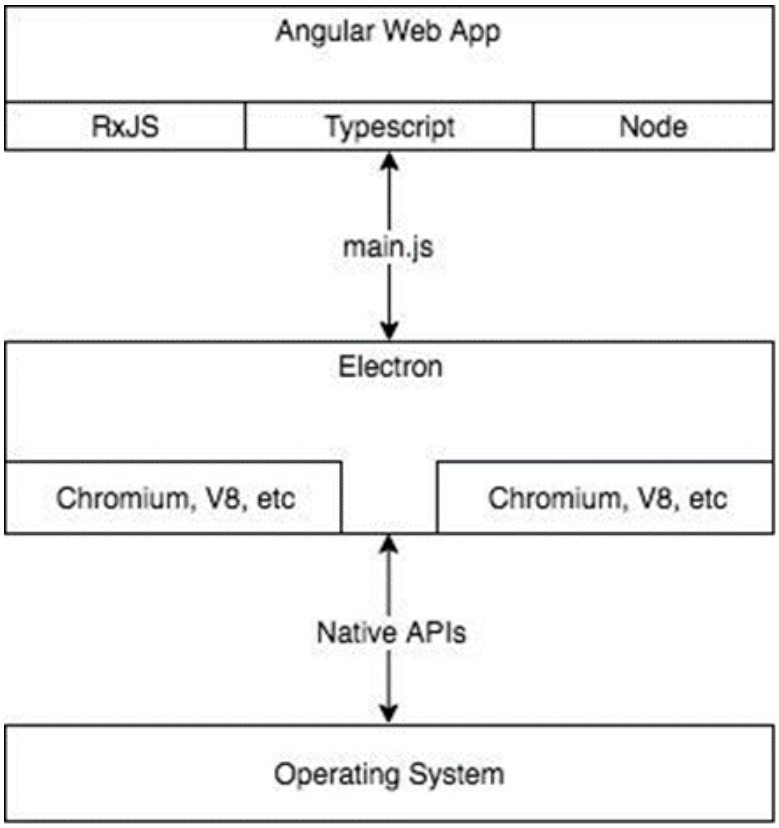

Рис. 2.7. Архітектура програми

Програмний застосунок для освітньої web-система містить у собі двох головних акторів – користувач системи і адміністратор системи. Головна відмінність у тому, що адміністратор має можливість створення курсів через спеціальну сторінку програмного застосунку. На рис. 2.8 наведено діграму варіантів використання.

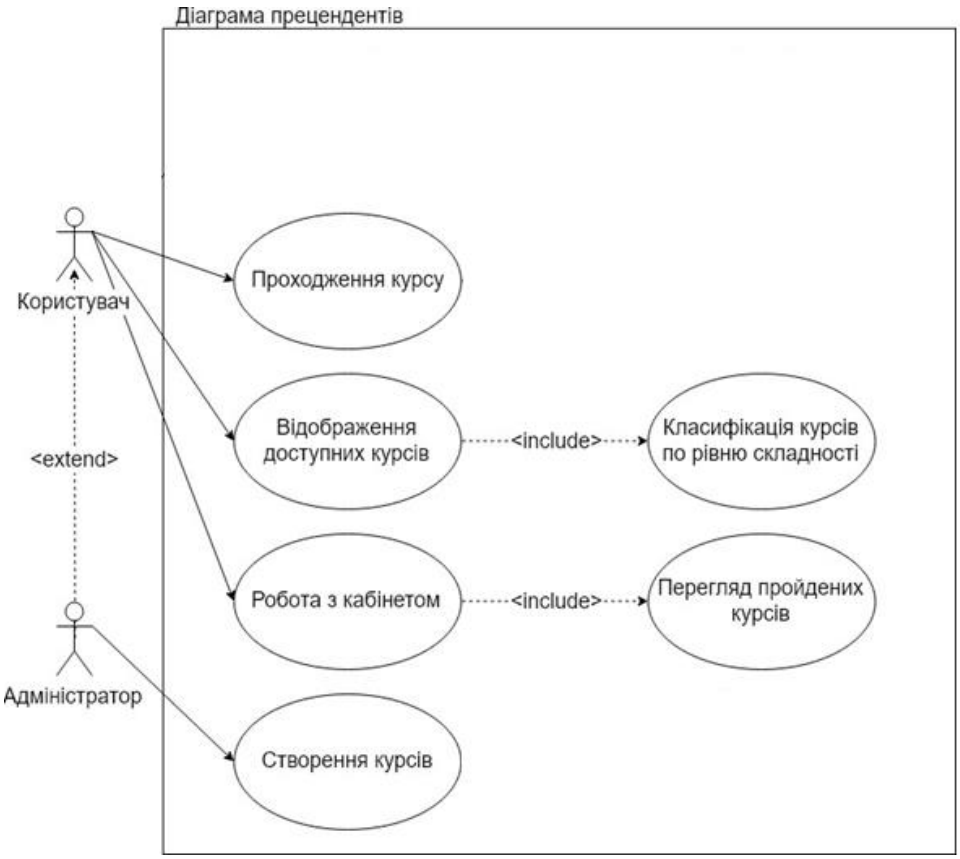

Рис. 2.8. Діграма варіантів використання

Розроблений Web-додаток буде здійснювати функції сполучної ланки між користувачем і БД:

- зберігання даних в базі даних (далі - БД) на сервері обумовлено наступними причинами: зберігання на сервері надійніше в порівнянні зі зберіганням на локальних машинах: до сервера обмежений як фізичний, так і програмний доступ, постійно виконується резервне копіювання даних;
- реляційна структура БД забезпечує більш швидкий доступ до пов'язаних даними;
- виключається небажане дублювання даних;
- можливість вибирати тільки ті дані, які необхідні в даний момент.

Надійність програми повинна забезпечуватися на рівні використовуваних апаратних і програмних засобів. Це досягається за необхідне контролем вхідних даних, їх обробкою і зберіганням.

Для Web-додатку передбачається установка сервера. На першому етапі можна обмежитися послугами хостингу або звичайним комп'ютером з заміною його

в майбутньому на повноцінний сервер (при необхідності). Необхідна потужність визначиться при експлуатації Web-додатку. Для початкового етапу вибираємо наступну конфігурацію сервера, наведену в таблиці 2.1.

Таблиця 2.1

Конфігурація серверу на початковому етапі

№	Назва компоненту	Кількість
1	Процесор сімества Celeron Gxxxx чи AMD Athlon	1
2	RAID масив рівня 1 з двох жорстких дисків по 100 GB	1
3	Модуль пам’яті 1024 MB DDR3	1
4	Материнська плата початкового рівня з неінтегрованим контролером LAN 1GB та вбудованим графічним контролером	1
5	Клавіатура, мишка, будь-який монітор	1
6	Корпус с блоком живлення FSP 350W ATX2.0	1

На обраному сервері необхідно встановити сервер Web-додатків, інтерпретатор, драйвер БД, а також встановити міжмережевий екран.

В такому випадку адміністратору мережі необхідно налаштувати відповідним чином DNS-сервер для правильної переадресації зовнішніх запитів до сервера Web-додатку. В такому випадку в якості клієнтських машин можуть виступати практично будь-які наявні ПК, за умови, що серед встановленого програмного забезпечення є браузер. Модернізувати апаратну частину цих ПК немає необхідності.

Для клієнта програмний додаток може працювати не тільки на ПК, але і на мобільних пристроях з встановленим браузером (бажано Google Chrome або Opera з включеним JavaScript). Необхідна наявність справного мережевого адаптера для здійснення підключення до мережі і доступу до сервера, наявність справних засобів введення інформації.

Мінімальні системні вимоги ідентичні вимогам до рекомендованих браузерам. Вивчивши предметну область і спроектувавши структуру Web-додатки,

можна приступити до наступного кроку - проектування структури бази дан них. В якості методу проектування БД обраний метод ER-діаграм.

Перш ніж розбирати дані зберігаються в базі WordPress, необхідно розгянути які є типи контенту. Існують такі стандартні типи контенту:

- записи (posts);
- сторінки (pages);
- користувальницькі типи записів (custom post types);
- вкладення (attachments);
- посилання (links);
- елементи меню (navigation menu items).

Ці типи контенту мають такі дані:

- категорії (categories);
- мітки (tags);
- користувальницькі таксономії (custom taxonomies and terms);
- метадані (post metadata).

Крім того існує типи контенту, що зберігаються в іншому вигляді:

- віджети (widgets);
- опції (options);
- користувачі (users);
- сайти для MU WordPress;
- нестандартний контент (hardcoded content), який додають деякі теми / плагіни;
- сторонній контент (third party content) (наприклад RSS).

Всі ці типи контенту зберігаються в таблицях бази даних або в файлах налаштувань тем / плагінів. Кожен тип може бути представлений як окремим записом в таблиці, так і її частиною. Крім, того вони можуть бути пов'язані з даними в інших таблицях. Наприклад, дані про записи пов'язані з даними про користувачів, так що WordPress знає, хто є автором, який записує.

WordPress використовує кілька взаємопов'язаних таблиць. Між ними встановлені зв'язку один до багатьох. Наприклад, до однієї сторінки може бути багато коментарів.

На рис. 2.9 представлена логічна модель спроектованої бази даних.

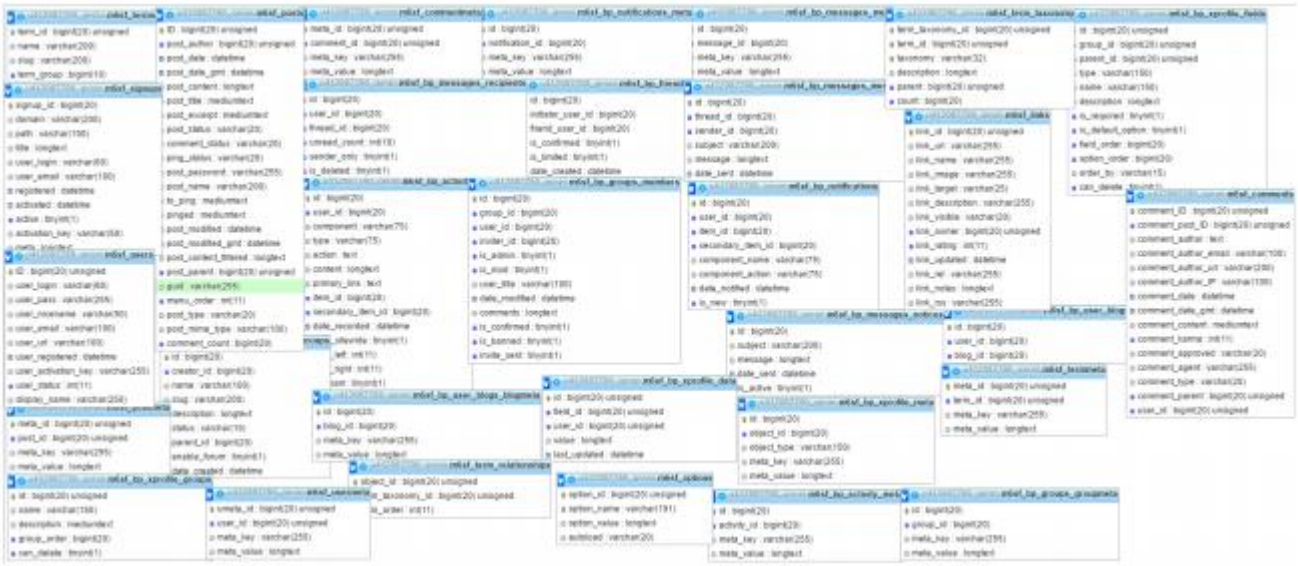

Рис. 2.9. Представлення таблиць

Вміст бази даних движка WordPress набагато більше - це тому, що вже є записи, сторінок, коментарів, і встановлені ті плагіни, котрі теж працюють на основі запитів до БД. Ось найпоширеніші WordPress настройки бази даних і запити, на яких, власне, і працює Web-додаток, побудований на WordPress:

- SELECT - вибрати рядки з таблиць;
- INSERT - додати рядки в таблицю;
- UPDATE - змінити рядки в заданій таблиці;
- DELETE - видалити рядки в заданій таблиці.

Це запити SQL (Structured Query Language - спеціальна мова для ма- ніпуляцій з даними).

Більшість таблиць пов'язані з однією або декількома іншими з допомогою одного поля. Це поле буде унікальним ідентифікатором для кожного запису (приклад post_id). У таблиці 3.2 детально представлені ці зв'язки.

Таблиця 2.2

Зв’язки між таблицями

№	Таблиця	Дані	Зв’язки з іншими таблицями
1	m6sf_posts	Записи, сторінки, вкладення, редакції, призначені для користувача записи	m6sf_postmeta через post_id, m6sf_term_relationships через post_id
2	m6sf_postmeta	Метадані записів та сторінок і т.д.	m6sf_posts через post_id
3	m6sf_comments	Коментарі	m6sf_posts через post_id
4	m6sf_commentmeta	Метадані коментарів	m6sf_comments через comment_id
5	m6sf_term_relationships	Зв'язки між таксономії і записами, сторінками і т.д.	m6sf_posts через post_id m6sf_term_taxonomy через term_taxonomy_id
6	m6sf_term_taxonomy	Таксономії (включаючи категорії і мітки)	m6sf_term_relationships через term_taxonomy_id
7	m6sf_terms	Ваші категорії, мітки і терміни призначених для користувача таксономій	m6sf_term_taxonomy черезterm_id
8	m6sf_links	Посилання в вашому блоці (як правило, зараз не використовується)	m6sf_term_relationships че-рез link_id
9	m6sf_users	Користувачі	m6sf_posts через post_author

Продовження таблиці 2.2

	Таблиця	Дані	Зв’язки з іншими таблицями
10	m6sf_user_meta	Метадані для кожного користувача	m6sf_users через user_id
11	m6sf_options	Опції і налаштування сайту (встановлюються в адмінки на сторінці налаштувань і в темах / плагинах)	Відсутні

Відзначимо деякі особливості цих таблиць:

- таблиці бази даних за замовчуванням мають префікс wp_. Потрібно його змінити (наприклад, при установці);
- таблиця m6sf_posts є найбільш важливо. Саме в ній зберігатися більшість даних;
- тільки одна таблиця не пов'язана з іншими - таблиця m6sf_options. У ній зберігаються дані про сайті і налаштуваннях WordPress, які не мають відношення до записів або користувачам;
- дві таблиці використовуються для зберігання даних про таксономії.

У таблицях m6sf_users і m6sf_comments дані не пов'язані. В налаштуваннях WordPress можна вказати, що тільки зареєстровані користувачі можуть залишити коментар. Не дивлячись на це, WordPress не зберігає зв'язку про коментарі і користувача, який їх відправив.

Таблиця 2.3

Типи контенту та таблиць

№	Тип контенту	Таблиця
1	Записи (posts)	m6sf_posts
2	Сторінки (pages)	m6sf_posts
3	Користувацькі типи записів (custom post types)	m6sf_posts
4	Вкладення (attachments)	m6sf_posts
5	Посиланя (links)	m6sf_links
6	Компоненти меню (navigation menu items)	m6sf_posts
7	Категорії(categories)	m6sf_terms
8	Метки (tags)	m6sf_terms
9	Користувацьці таксономії (custom taxonomies)	m6sf_term_taxonomy
10	Терміни користувацьких таксономій (custom terms)	m6sf_terms
11	Метадані (post metadata)	m6sf_postmeta
12	Віджети (widgets)	m6sf_options
13	Опції (options)	m6sf_options
14	Користувачі(users)	m6sf_users
15	Нестандартний контент (hardcoded content)	m6sf_posts, m6sf_options, файли тем/плагинів
16	Стороній контент (third party content)	m6sf_posts, m6sf_options, файли тем/плагинів

Відзначимо, що не всі таблиці використовуються в переліку. Так відбувається тому, що деякі з них використовуються для зберігання метаданих. Інші використовуються для зберігання зв'язків.

2.4. Засоби розробки бази даних

СУБД - це комплекс мовних і програмних засобів, призначений для створення, ведення і сумісного використання БД багатьма користувачами. Зазвичай СУБД розрізняють по використовуваної моделі даних. Так, СУБД, засновані на використанні реляційної моделі даних, називають реляційними СУБД.

Критерії для вибору СУБД:

• стабільність;

• функціональність;

• вартість.

Розглянемо наступні СУБД:

• Microsoft SQL Server express;

• MySQL;

• PHPMyAdmin.

Microsoft SQL Server Express - система керування базами даних (СУРБД), розроблена корпорацією Microsoft. Основний використовуваний мову запитів - Transact-SQL, створений спільно Microsoft та Sybase. Transact-SQL є реалізацією стандарту ANSI / ISO по структурованого мови запитів (SQL) з розширеннями. Використовується для роботи з базами даних розміром від персональних до великих баз даних масштабу підприємства; конкурує з іншими СУБД в цьому сегменті ринку.

SQL Server Express включає сховище об'ємом 10 ГБ на кожну базу даних, зручні функції резервного копіювання та відновлення, а також сумісний з усіма випусками баз даних SQL Server і Microsoft Azure SQL. Поширюється на безкоштовній основі.

MySQL - вільна система керування базами даних. Розробка та підтримка сайта MySQL здійснює корпорація Oracle, що отримала права на торговельну марку разом з поглиненої Sun Microsystems, яка раніше придбала шведську компанію MySQL AB. Продукт поширюється як під GNU General Public License, так і під власною комерційною ліцензією. Окрім цього, розробники створюють

функціональність за замовленням ліцензійних користувачів. Саме завдяки такому замовленню майже в найраніших версіях з'явився механізм реплікації.

Гнучкість СУБД MySQL забезпечується підтримкою великої кількості типів таблиць: користувачі можуть вибрати як таблиці типу MyISAM, що підтримують повнотекстовий пошук, так і таблиці InnoDB, що підтримують транзакцыъ на рівні окремих записів. Більш того, СУБД MySQL поставляється із спеціальним типом таблиць EXAMPLE, що демонструє принципи створення нових типів таблиць. Завдяки відкритій архітектурі і GPL-ліцензуванню, в СУБД MySQL постійно з'являються нові типи таблиць.

MySQL має подвійне ліцензування. MySQL може поширюватися відповідно до умов ліцензії GPL. Однак за умовами GPL, якщо яка-небудь програма включає вихідні коди MySQL, то вона теж повинна поширюватися на умовах ліцензії GPL. Це може розходитися з планами розробників, які не бажають відкривати вихідні тексти своїх програм. Для таких випадків передбачена комерційна ліцензія, яка також забезпечує якісну сервісну підтримку. Поширюється на безкоштовній основі.

На підставі отриманих даних був обраний для розробки програми MySQL, так як вона повно функціональна і безкоштовна, а так само не має будь-яких обмежень в розмірі бази даних. Так само для даної реляційної системи існує безліч додатків спрощують адміністрування бази даних. Найчастіше, це виявляється PhpMyAdmin. PHPMyAdmin - безкоштовний додаток з відкритим кодом, призначений для адміністрування СУБД MySQL. PHPMyAdmin є веб-інтерфейс за допомогою якого можна адмініструвати сервер MySQL, запускати команди і переглядати вміст таблиць і БД через браузер.

Даний додаток є дуже популярним через його плюси: можливість керувати СУБД MySQL без безпосереднього введення SQL команд, як панель управління PHPMyAdmin надає можливість адміністрування виділених БД, інтенсивний розвиток, можливість інтегрувати PHPMyAdmin у власні розробки завдяки ліцензії GNU General Public License та інші можливості.

На підставі цієї інформації в якості Web-інтерфейсу для бази даних MySQL був обраний PhpMyAdmin так, як даний Web-додаток дуже популярно і поширюється абсолютно безкоштовно.

Web-додаток складається з двох рівноцінних частин: серверної і клієнтської. Можливо, використовувати одну і ту ж мову програмування у всьому додатку, проте це є самим неоптимальним варіантом.

Для серверної частини буде використовуватися PHP. PHP (англ. PHP: Hypertext Preprocessor - «PHP: препроцесор гіпертексту») - скриптова мова програмування загального призначення, інтенсивно застосовується для розробки Web-додатків. В даний час підтримується переважною більшістю хостинг-провайдерів і є одним з лідерів серед мов програмування, що застосовуються для створення динамічних Web-сайтів.

Для клієнтської частини будуть використовуватися мови, що наведені нижче.

HTML

HTML (від англ. HyperText Markup Language - «мова розмітки гіпертексту») - стандартна мова розмітки документів у Всесвітній павутині.

CSS

CSS (від англ. Cascading Style Sheets - каскадні таблиці стилів) - формальна мова опису зовнішнього вигляду документа, написаного з використанням мови розмітки.

JavaScript

JavaScript - об'єктно-орієнтована скриптова мова програмування. JavaScript підтримується всіма існуючими браузерами і є стандартом де-факто для сучасних інтерактивних Web-додатків.

Node.js

Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Засновником платформи є Раян Дал (Ryan Dahl). Якщо раніше Javascript застосовувався для обробки даних в браузері користувача, то node.js надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їхнього виконання. Платформа

Node.js перетворила JavaScript на мову загального використання з великою спільнотою розробників.

2.5. Висновки до розділу

В даному розділі були розглянуті наступні важливі питання:

1. проектування загальної структури програми;
2. описано принцип взаємодії компонентів, додатки, виділені основні потоки даних, якими компоненти обмінюються між собою;
3. побудовано контекстна і деталізують діаграми потоків даних;
4. вибір засобів реалізації Web-додатки, обрані програмні інструменти для практичної реалізації бази даних і компонентів розробленої програми з обгрунтуванням прийнятих рішень;
5. розробка керівництва користувача і опис роботи програми.

Детально викладена послідовність дій користувача, описані ключові можливості програми.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ В ПРОГРАМНОМУ СЕРЕДОВИЩІ

3.1. Результати роботи програми

Інтерфейс користувача програми розроблявся з урахуванням вимог простоти, зручності і комфорту. Початкова сторінка додатка є інформацію про останні новинних записах в Web-додатку (рис. 3.1).

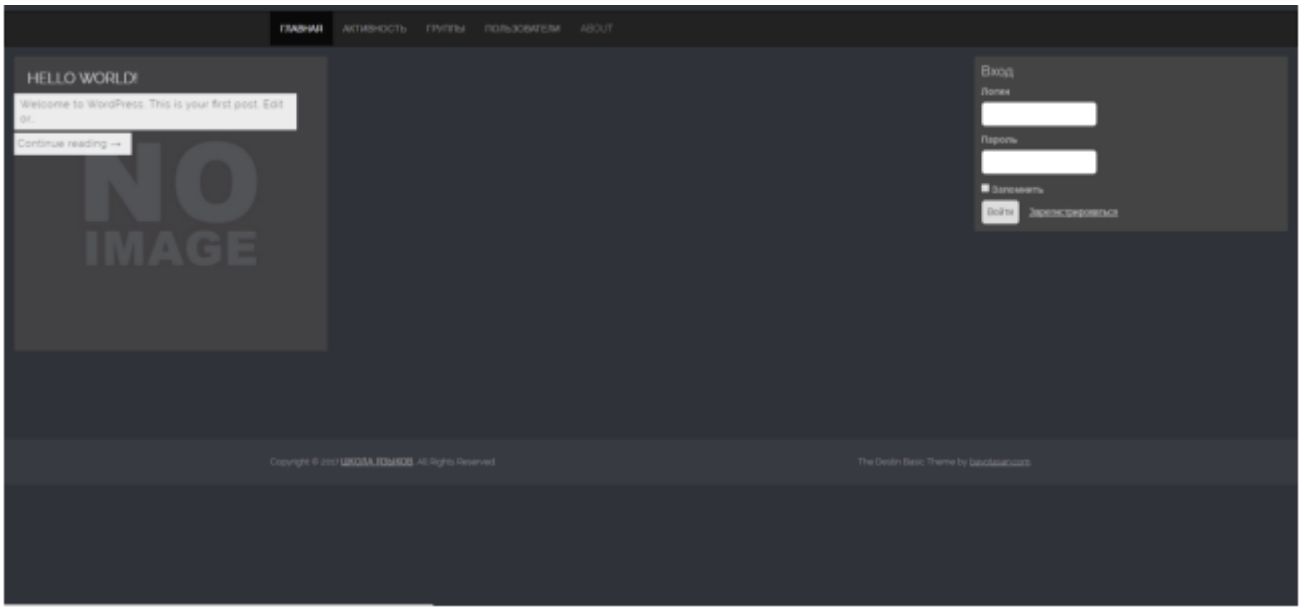

Рис. 3.1. Інтерфейс початкової сторінки

Так само на головній сторінці присутній форма «Авторизации», по якій користувачі можуть авторизуватися, якщо вже мають обліковий запис, або ж зареєструватися (рис. 3.2).

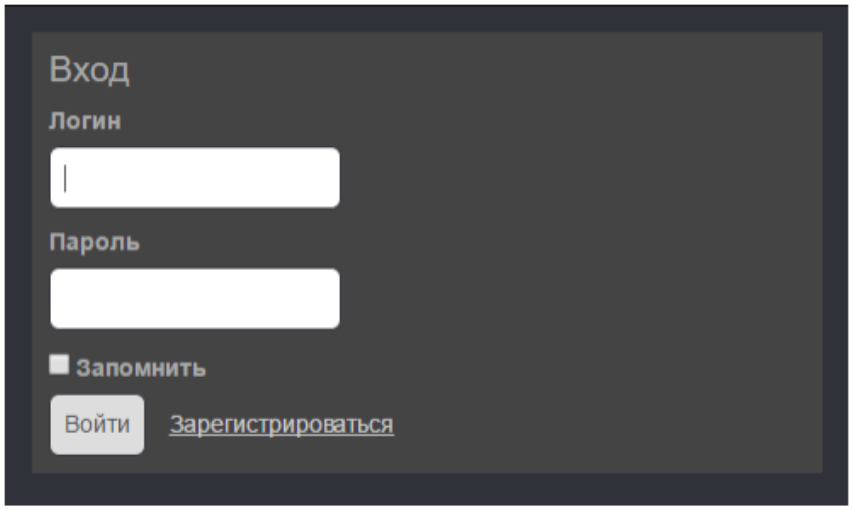

Рис. 3.2. Вікно авторизації

Якщо новий користувач не має бажання реєструватися, то він може просто вивчити Інтернет-ресурс, скориставшись панеллю навігації розташованої зверху маючи при цьому тільки доступ «режим перегляду» (рис. 3.3).

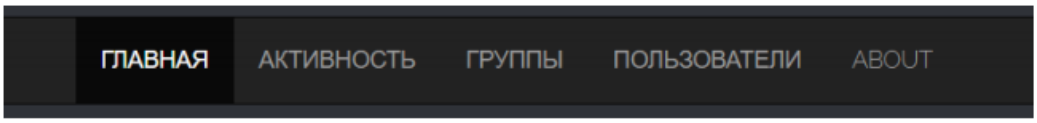

Рис. 3.3. Панель навігації

На сторінці «Про сайт» розташована інформація про організацію, самому сайті і контактні дані (рис. 3.4.). Так само присутній форма для заповнення за допомогою, якої можна залишити відгук про сайт і організації, в загальному.

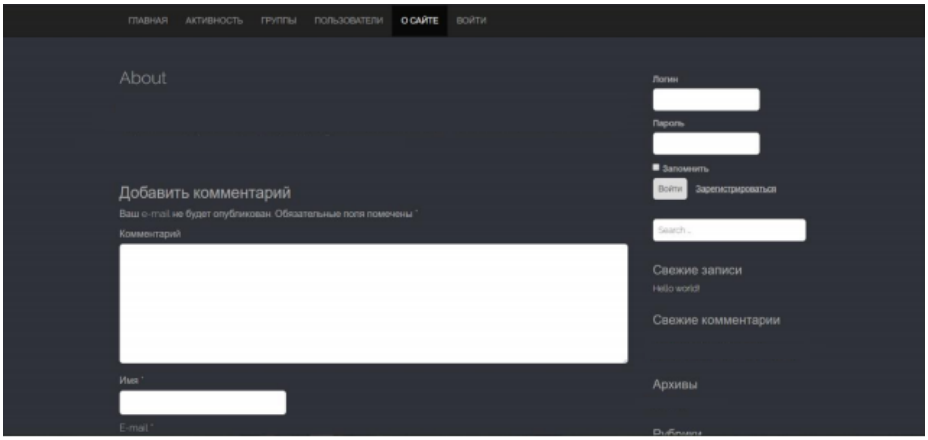

Рис. 3.4. Вміст сторінки «Про сайт»

Якщо Ви увійшли до системи, то йому додатково стає доступні деякі функції. Такі як перегляд свого профілю (рис. 3.5). На сторінку профілю можна потрапити, якщо клікнути курсором на ім'я користувача.

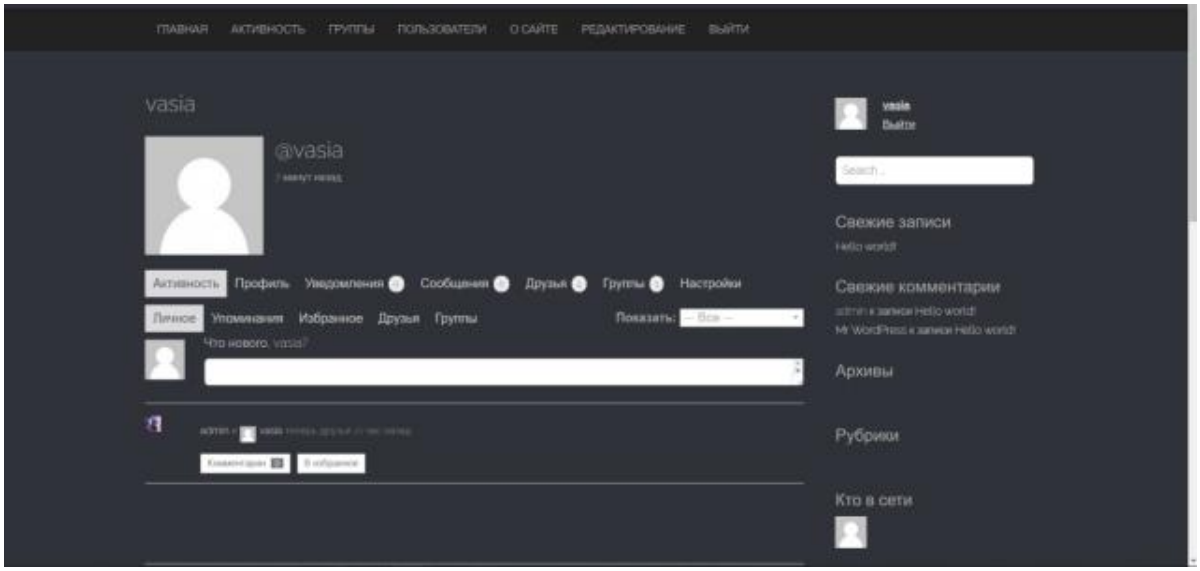

Рис. 3.5. Вміст сторінки «Профіль користувача»

Метою даної випускної кваліфікаційної роботи є розробка Web-додатки надає інформаційно-навчальний матеріал і підтримує зв'язок між користувачами.

Зниження трудових витрат дозволить зменшити і фінансові витрати, що приведе до загального збільшення продуктивності і економії у процесі впровадження на інформаціних порталах. Цей дотаток може бути впроваджений у будь-який навчальний заклад, або приватний заклад, що надає послуги освітнього характеру.

В даному розділі буде розрахований економічний ефект від використання Web-додатку.

Основним завданням цього розділу є визначення величини витрат на проведення досліджень, собівартість для визначення економічного ефекту від використання в суспільному виробництві основних і супутніх результатів, одержуваних при вирішенні поставленої технічної задачі в цій випускній кваліфікаційнії роботі. Оцінка ефективності прийнятого науково-технічного рішення повинна враховувати всі необхідні витрати для цього буде потрібно

провести ряд необхідних розрахунків. Вартість витрат визначається зі списків цін в мережевих магазинах, і представлена в таблиці 3.1.

Таблиця 3.1

Вартість розробки

№	Назва	Кількість	Вартість
1	HP Pavilion 15-n005sr + Windows 10	1	25000
2	Apache HTTP Server	1	0
3	СУБД MySql + phpMyAdmin	1	0
4	ПЗ для роботи	1	0
ЗАГАЛОМ		-	25000

Таблиця 3.2

Загальні витрати на розбобку та впровадження додатку

№	Витратри	Сумма, грн.
1	Придбання, утримання та експлуатація програмного і апаратного забезпечення	25000
2	Витрати на основну заробітню плату	15000
3	Витрати на додаткову заробітню плату	5000
4	Відрахування на соціальні потреби	4000
5	Амортизаційні відрахування	234,4
ЗАГАЛОМ		49234,4

3.2. Висновки до розділу

У розділі було розглянуто процес розробки веб застосунку. Створення інтерфейсу та прораховано економічні затрати на розробку додатку.

Через відсутність готового дизайну, і відсутність у мене необхідної експертності для цього, щоб додаток було готовий до впровадження замовнику, потрібно ще більше часу, однак загальний результат позитивний. Узгодженість на різних

платформах і непослідовна підтримка CSS на різних платформах є однією з основних проблем при створенні міжплатформного, призначеного для користувача, інтерфейсу.

Основним завданням цього розділу є визначення величини витрат на проведення досліджень, собівартість для визначення економічного ефекту від використання в суспільному виробництві основних і супутніх результатів, одержуваних при вирішенні поставленої технічної задачі в цій випускній кваліфікаційнії роботі. Оцінка ефективності прийнятого науково-технічного рішення повинна враховувати всі необхідні витрати для цього буде потрібно провести ряд необхідних розрахунків.

ВИСНОВКИ

У дипломній роботі було проведено аналіз веб систем. Наведено весь спектр засобів розробки веб застосунків. Розглянуто перспективи розвитку веб технологій.

Вплив глобальної комп'ютерної мережі Internet на сучасний світ не має історичних аналогів. Його сьогоднішній день - це початок епохи електронного заповнення усіх сфер людського життя, це основа нової філософії і нового способу мишлення.

Web-технології повністю перевернули уявлення про роботу з інформацією, та й з комп'ютером взагалі. Виявилося, що традиційні параметри розвитку обчислювальної техніки - продуктивність, пропускна здатність, ємність запам'ятовуючих пристроїв - не враховували головного "вузького місця" системи - інтерфейсу з людиною. Застарілий механізм взаємодії людини з інформаційною системою стримував впровадження нових технологій і зменшував вигоду від їх застосування. І тільки коли інтерфейс між людиною і комп'ютером був спрощений до природності сприйняття звичайною людиною, послідував безпрецедентний вибух інтересу до можливостей обчислювальної техніки.

Сучасні технології інтегруються в різні сфери нашого життя щодня. Одна з таких сфер – освіта, що розвивається щодня, розробляються нові методи для покращення рівню освіти. Саме тому інтеграція сучасних технологій в освітній процес сприяє зміні підходу до навчання, збільшенню об’єму поглинутих знань, використанню нові технології для самореалізації у не доступних раніше сферах.

У роботі актуалізується проблема ефективності використання сучасних WEB-технологій при реалізації WEB-інтерфейсу користувача, а також проблема організації навчального процесу з використанням новітніх технологій. Пропонований варіант рішення дозволяє успішно інтегрувати стек сучасних WEB-технологій та застосувати його в розробці програмного комплексу для провдення занять у ігровій формі, сприяє підвищенню зацікавленості учнів в навчальному процесі та автоматизує процес процес проведення котрольних та заліків.

Побудований додаток повністю задовольняє всі представлені до нього вимоги і виконує всі бажані функції. Тестування виявило мінімальні проблеми з відображенням на різних платформах.

Через відсутність готового дизайну, і відсутність у мене необхідної експертності для цього, щоб додаток було готовий до впровадження замовнику, потрібно ще більше часу, однак загальний результат позитивний. Узгодженість на різних платформах і непослідовна підтримка CSS на різних платформах є однією з основних проблем при створенні міжплатформного, призначеного для користувача, інтерфейсу.

СПИСОК БІБЛІОГРАФИЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Иванова, Г.С. Технология программирования: учебник/ Г.С. Иванова – М.: КНОРУС, 2011. – 336 с.
2. Астахова, И.Ф. SQL в примерах и задачах: учебное пособие/ И.Ф. Астахова, А.П. Толстобров, В.М. Мельников – Минск: Новое знание, 2002. – 176 с.
3. Дейт, К. Дж. Введение в системы баз данных/Кристофер Дейт: пер. с англ. – 8-е изд. – М.: Издательский дом «Вильямс», 2006. – 1328 с.
4. Поддубный, А. Расчет экономического эффекта от внедрения системы автоматизации. [Електроний ресурс]. — Режим доступу: http://www.antegra.ru/news/experts/_det-experts/4/ (дата звернення 28.05.2021р.) – Назва з екрана.
5. Орлова, И.В. Экономико-математические методы и модели: компьютерное моделирование: Учебное пособие / И.В. Орлова. – М.: Вузовский учебник, НИЦ ИНФРА-М, 2013. - 389 с.
6. А.И.Дмитриенко. Информационная безопасность ERP-систем [Електроний ресурс] — Режим доступу: http://cyberleninka.ru/ article/n/informatsionnaya-bezopasnost-erp-sistem-1 (дата звернення 25.05.2021р.) – Назва з екрана .
7. Myeongju Ji Sungryong Kim Yongjin Park. Mobile device management system with portable devices // Consumer Electronics (ISCE), 2015 IEEE International Symposium on. — 11-16 May 2015., Seoul, Korea .
8. Адам Фримен — ASP.NET Core MVC с примерами на C# для профессионалов [Електронний ресурс] — Режим доступу: https://bit.ly/2JbSgHe (дата звернення 28.05.2021р.) – Назва з екрана.
9. Джеймс Чамберс — ASP.NET Core. Разработка приложений [Електронний ресурс] — Режим доступу: https://bit.ly/2AliYL2 (дата звернення 03.06.2021р.) – Назва з екрана.
10. Markus Egger — MVVM Survival Guide for Enterprise Architectures in Silverlight and WPF [Електронний ресурс]. — Режим доступу: https://www.packtpub.com/application-development/mvvm-survival-guide-

enterprisearchitectures-silverlight-and-wpf (дата звернення 04.06.2021р.) – Назва з екрана.

11. Anthony Gore — Full-Stack Javascript [Електронний ресурс] — Режим доступу: https://bit.ly/2OEODzR (дата звернення 01.06.2021р.) – Назва з екрана.

12. Kyle Simpson - You Don't Know JS: Up & Going [Електронний ресурс] - Режим доступу: https://www.ebooks.com/en-ua/1993212/you-don-t-know-js-upgoing/simpson-kyle (дата звернення 28.05.2021р.) – Назва з екрана.

13. Болье А. — Learning NoSQL [Електронний ресурс] — Режим доступу: http://shop.oreilly.com/product/9780596007270.do (дата звернення 15.05.2021р.) – Назва з екрана.

ДОДАТКИ

ДОДАТОК А

Лістинг програми

src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import 'antd/dist/antd.css';
import './normalize.css';
import { Provider } from 'react-redux';
import App from './view/App';
import registerServiceWorker from './registerServiceWorker';
import store from './store';
ReactDOM.render(
 <Provider store={store}>
 <App />
 </Provider>, document.getElementById('root'),
);
registerServiceWorker();
```

src/App.js

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { history } from 'store';
import { bindActionCreators } from 'redux';
import { Redirect, Switch, Route } from 'react-router-dom';
import { ConnectedRouter } from 'connected-react-router';
import { /* Header, */ Sider, Spinner } from 'components';
import { Layout } from 'antd';
import 'assets/fonts/fonts.css';
import { checkIsUserLoggedIn } from '../redux/user/actions';
```

```
import Dashboard from './Dashboard';
import Login from './Login';
import Registration from './Registration';
import { getCourseList, getRoadmap } from '../redux/courses/actions';
import Roadmap from './Roadmap';
import UserPage from './UserPage';
import CoursePage from './CoursePage';
import CoursesList from './CoursesList';
import LessonPage from './LessonPage';
import CreateCoursePage from './CreateCoursePage';
const ProtectedRoute = ({
 isUserLoggedIn, path, component, exact,
}) => (
 isUserLoggedIn
 ? <Route exact={exact} path={path} component={component} />
 : <Redirect to="/login" />
);
const MainLayout = () => (
 <Layout>
 <Sider />
 <Layout>
 {/* <Header style={{ background: '#fff', padding: 0 }} /> */}
 <Layout.Content style={{ margin: '0 16px' }}>
 <Switch>
 <Route exact path="/" component={Dashboard} />
 <Route path="/courses" component={CoursesList} />
 <Route path="/roadmap" component={Roadmap} />
 <Route path="/course/:id/lesson/:lessonId" component={LessonPage} />
 <Route path="/course/:id" component={CoursePage} />
 <Route path="/create-course" component={CreateCoursePage} />
```

```
 <Route path="/profile" component={UserPage} />
 <Redirect to="/" />
 </Switch>
 </Layout.Content>
 </Layout>
 </Layout>
);
class App extends Component {
 constructor(props) {
 super(props);
 this.state = {};
 this.initReduxPart();
 }
 initReduxPart = () => {
 const { actions: { checkIsUserLoggedIn, getCourseList, getRoadmap } } =
this.props;
 checkIsUserLoggedIn();
 getCourseList();
 getRoadmap();
 };
render() {
 const { isUserLoggedIn, initialPending } = this.props;
if (initialPending) return <Spinner />;
 return (
 <ConnectedRouter history={history}>
 <Layout style={{ minHeight: '100vh' }}>
 <Switch>
 <Route exact path="/login" component={Login} />
 <Route exact path="/registration" component={Registration} />
```

```
<ProtectedRoute isUserLoggedIn={isUserLoggedIn} path="/" component={MainLayout} />
 </Switch>
 </Layout>
 </ConnectedRouter>);
 }}
function mapStateToProps(state) {
 return {
 isUserLoggedIn: state.user.isUserLoggedIn,
 initialPending: state.user.initialPending, };}
function mapDispatchToProps(dispatch) {
return {
 actions: bindActionCreators(
{ checkIsUserLoggedIn,
 getCourseList,
 getRoadmap,
 },  dispatch,
 ),  };}
export default connect(mapStateToProps, mapDispatchToProps)(App);
src/registerServiceWorker.js
const isLocalhost = Boolean(
 window.location.hostname === 'localhost' ||
 window.location.hostname === '[::1]' ||
 window.location.hostname.match(
 /^127(?:\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3}$/));
export default function register() {
 if (process.env.NODE_ENV === 'production' && 'serviceWorker' in navigator)
{
 // The URL constructor is available in all browsers that support SW.
 const publicUrl = new URL(process.env.PUBLIC_URL, window.location);
```

```
 if (publicUrl.origin !== window.location.origin) {
 return; }
window.addEventListener('load', () => {
 const swUrl = `${process.env.PUBLIC_URL}/service-worker.js`;
if (isLocalhost) {
 checkValidServiceWorker(swUrl);
 navigator.serviceWorker.ready.then(() => {
 console.log(
 'This web app is being served cache-first by a service ' +
 'worker. To learn more, visit https://goo.gl/SC7cgQ'
 ); });
 } else {
 registerValidSW(swUrl);
 }}); }}
function registerValidSW(swUrl) {
 navigator.serviceWorker
 .register(swUrl)
 .then(registration => {
 registration.onupdatefound = () => {
 const installingWorker = registration.installing;
 installingWorker.onstatechange = () => {
 if (installingWorker.state === 'installed') {
 if (navigator.serviceWorker.controller) {
 console.log('New content is available; please refresh.');
} else {
 console.log('Content is cached for offline use.');  } }}; }; })
 .catch(error => {
 console.error('Error during service worker registration:', error);
 }); }
function checkValidServiceWorker(swUrl) {
```

```
 fetch(swUrl)
 .then(response => {
 if (
 response.status === 404 ||
 response.headers.get('content-type').indexOf('javascript') === -1
 ) {
 navigator.serviceWorker.ready.then(registration => {
 registration.unregister().then(() => {
 window.location.reload();
 }); });
 } else {
 registerValidSW(swUrl);  } })
.catch(() => {
 console.log(
'No internet connection found. App is running in offline mode.'  ); });}
export function unregister() {
 if ('serviceWorker' in navigator) {
 navigator.serviceWorker.ready.then(registration => {
 registration.unregister();});}}
src/view/CoursePage/index.js
import { connect } from 'react-redux';
import { withRouter } from 'react-router-dom';
import { bindActionCreators } from 'redux';
import { logout } from 'redux/user/actions';
import { getCourseData } from 'redux/courses/actions';
import CoursePage from './CoursePage';
function mapStateToProps(state) {
 return {
 courses: state.courses.courseList,
 courseData: state.courses.courseData, };}
```

```
function mapDispatchToProps(dispatch) {
 return {
actions: bindActionCreators(
 { logout,
getCourseData, },
 dispatch, ), };}
export default withRouter(connect(mapStateToProps,
mapDispatchToProps)(CoursePage));
src/view/CreateCoursePage/index.js
import { connect } from 'react-redux';
import { withRouter } from 'react-router-dom';
import { bindActionCreators } from 'redux';
import { getSensorList } from 'redux/sensor/actions';
import { showNotification } from 'redux/notification/actions';
import CreateCoursePage from './CreateCoursePage';
function mapStateToProps(state) {
 return {
 sensors: state.sensor.sensors,
 actionTypes: state.sensor.actionTypes,
 arduinos: state.sensor.arduinos,
 places: state.sensor.places,
 };}
function mapDispatchToProps(dispatch) {
 return {
actions: bindActionCreators(
 { getSensorList,
 showNotification, },
 dispatch,
 ), };}
export default withRouter(connect(mapStateToProps,
```

```
mapDispatchToProps)(CreateCoursePage));
src/view/Dashboard/index.js
import { connect } from 'react-redux';
import { withRouter } from 'react-router-dom';
import { bindActionCreators } from 'redux';
import { logout } from 'redux/user/actions';
import Dashboard from './Dashboard';
function mapStateToProps(state) {
 return {
 user: state.user,
 };}
function mapDispatchToProps(dispatch) {
return {
 actions: bindActionCreators(
{  logout,},
 dispatch,  ), };}
export default withRouter(connect(mapStateToProps,
mapDispatchToProps)(Dashboard));
src/view/LessonPage/index.js
import { connect } from 'react-redux';
import { withRouter } from 'react-router-dom';
import { bindActionCreators } from 'redux';
import { logout, trackActivity, completeLesson } from 'redux/user/actions';
import { getCourseData, getLessonData } from 'redux/courses/actions';
import { showNotification } from 'redux/notification/actions';
import LessonPage from './LessonPage';
function mapStateToProps(state) {
 const activeSection = state.courses.courseData.sections
 .find(el => el.lessons.indexOf(state.courses.lessonData['_id'] !== -1)) || {};
 return {
```

```
 course: state.courses.courseData.course || {},
 lessonData: state.courses.lessonData,
courseData: state.courses.courseData,
activeSection,
activeSectionIndex: state.courses.courseData.sections.map(el => el['_id'])
 .indexOf(activeSection['_id']),
 user: state.user, };}
function mapDispatchToProps(dispatch) {
 return {
 actions: bindActionCreators(
 { logout,
 getCourseData,
 getLessonData,
 showNotification,
 trackActivity,
 completeLesson,},
 dispatch, ),};}
export default withRouter(connect(mapStateToProps,
mapDispatchToProps)(LessonPage));
src/view/Login/index.js
import { connect } from 'react-redux';
import { withRouter } from 'react-router-dom';
import { bindActionCreators } from 'redux';
import { login } from 'redux/user/actions';
import LoginPage from './Login';
function mapStateToProps(state) {
 return { isUserLoggedIn: state.user.isUserLoggedIn,
 };}
function mapDispatchToProps(dispatch) {
 return {
```

```
actions: bindActionCreators(
 { login, },
 dispatch,), };}
export default withRouter(connect(mapStateToProps,
mapDispatchToProps)(LoginPage));
src/view/Registration/index.js
import { connect } from 'react-redux';
import { withRouter } from 'react-router-dom';
import { bindActionCreators } from 'redux';
import { registration } from 'redux/user/actions';
import RegistrationPage from './Registration';
function mapStateToProps(state) {
return {
isUserLoggedIn: state.user.isUserLoggedIn, };}
function mapDispatchToProps(dispatch) {
 return {
 actions: bindActionCreators(
 { registration, },
 dispatch, ), };}
export default withRouter(connect(mapStateToProps,
mapDispatchToProps)(RegistrationPage));
src/view/Roadmap/index.js
import { connect } from 'react-redux';
import { withRouter } from 'react-router-dom';
import { bindActionCreators } from 'redux';
import { logout } from 'redux/user/actions';
import Roadmap from './Roadmap';
function mapStateToProps(state) {
 return {
 roadmap: state.courses.roadmap, };}
```

```
function mapDispatchToProps(dispatch) {
 return {
actions: bindActionCreators(
 { logout, },
dispatch,), };}
export default withRouter(connect(mapStateToProps,
mapDispatchToProps)(Roadmap));
src/view/UserPage/index.js
import { connect } from 'react-redux';
import { withRouter } from 'react-router-dom';
import { bindActionCreators } from 'redux';
import { logout, updateUser } from 'redux/user/actions';
import UserPage from './UserPage';
function mapStateToProps(state) {
 return {
 user: state.user, };}
function mapDispatchToProps(dispatch) {
 return {
 actions: bindActionCreators(
 { logout, updateUser,},
 dispatch,  ), }; }
export default withRouter(connect(mapStateToProps,
mapDispatchToProps)(UserPage));
```